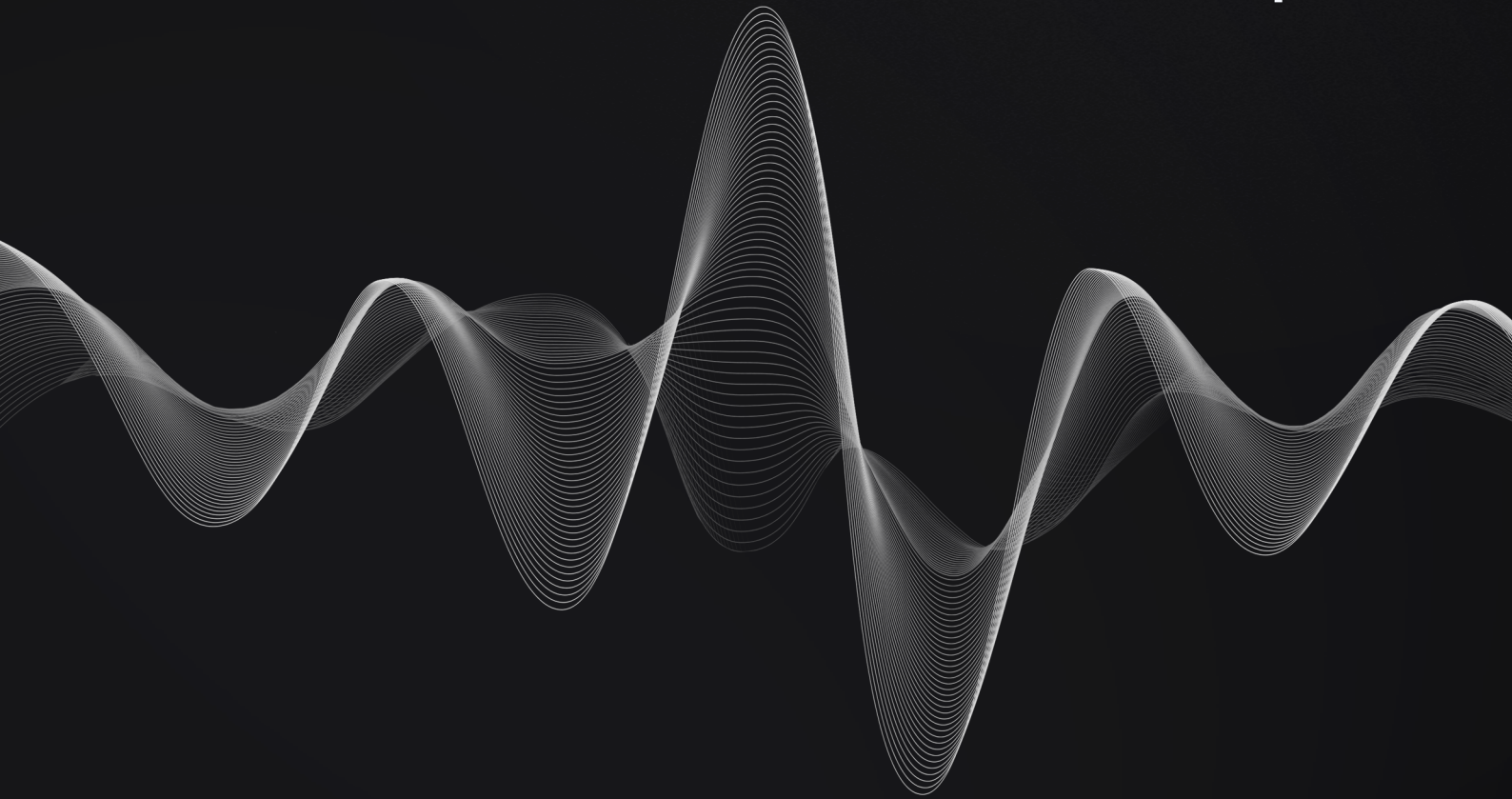


DEIN Finance

DEIN Smart Contract Audit Report









Document Control

PUBLIC

FINAL(v2.1)

Audit_Report_DEIN-INS_FINAL_21

Feb 18, 2025		v0.1	Luis Arroyo: Initial draft
Feb 19, 2025		v0.2	Luis Arroyo: Added findings
Feb 20, 2025		v1.0	Charles Dray: Approved
Apr 15, 2025		v1.1	Luis Arroyo: Reviewed findings
Apr 16, 2025		v2.0	Charles Dray: Finalized
Apr 16, 2025		v2.1	Charles Dray: Published

Points of Contact	Casper Skorka Charles Dray	Dein Finance Resonance	casper@dein.fi charles@resonance.security
--------------------------	-------------------------------	---------------------------	--

Testing Team	Luis Arroyo João Simões Michał Bazyli	Resonance Resonance Resonance	luis.arroyo@resonance.security joao@resonance.security michal@resonance.security
---------------------	---	-------------------------------------	--



Copyright and Disclaimer

© 2025 Resonance Security, Inc. All rights reserved.

The information in this report is considered confidential and proprietary by Resonance and is licensed to the recipient solely under the terms of the project statement of work. Reproduction or distribution, in whole or in part, is strictly prohibited without the express written permission of Resonance.

All activities performed by Resonance in connection with this project were carried out in accordance with the project statement of work and agreed-upon project plan. It's important to note that security assessments are time-limited and may depend on information provided by the client, its affiliates, or partners. As such, the findings documented in this report should not be considered a comprehensive list of all security issues, flaws, or defects in the target system or codebase.

Furthermore, it is hereby assumed that all of the risks in electing not to remedy the security issues identified henceforth are sole responsibility of the respective client. The acknowledgement and understanding of the risks which may arise due to failure to remedy the described security issues, waives and releases any claims against Resonance, now known or hereafter known, on account of damage or financial loss.

Contents

1 Document Control	2
Copyright and Disclaimer	2
2 Executive Summary	4
System Overview	4
Repository Coverage and Quality.....	4
3 Target	6
4 Methodology	7
Severity Rating.....	8
Repository Coverage and Quality Rating.....	9
5 Findings	10
Restake Rewards Allow Extra Benefits In Less Time.....	11
permit() Function Could Be Frontrun	12
Hardcoded blocks_per_day May Fail in L2 Chains or Consensus Upgrades	13
Missing Limits In setMinimumInsuranceCost()	14
Fee On Transfer Tokens May Not Work Properly	15
epoch in buyPolicy() Does Not Follow Documentation.....	16
Execution at Deadlines Should Be Allowed.....	17
TWAP Oracle Update Mechanism Cannot Be Left To Users	18
Potential TWAP Manipulation if DEIN/ETH Pool Has Low Liquidity.....	19
evidenceURI Could Contain Malicious Payloads	20
Non-external Function Names Should Begin With an Underscore	21
uint Should Not Be Initialized With 0	22
A Proof of Concepts	23

Executive Summary

DEIN Finance contracted the services of Resonance to conduct a comprehensive security audit of their smart contracts between January 23, 2025 and February 20, 2025. The primary objective of the assessment was to identify any potential security vulnerabilities and ensure the correct functioning of smart contract operations.

During the engagement, Resonance allocated 2 engineers to perform the security review. The engineers, including an accomplished professional with extensive proficiency in blockchain and smart-contract security, encompassing specialized skills in advanced penetration testing, and in-depth knowledge of multiple blockchain protocols, devoted 20 days to the project. The project's test targets, overview, and coverage details are available throughout the next sections of the report.

The ultimate goal of the audit was to provide DEIN Finance with a detailed summary of the findings, including any identified vulnerabilities, and recommendations to mitigate any discovered risks. The results of the audit are presented in detail further below.



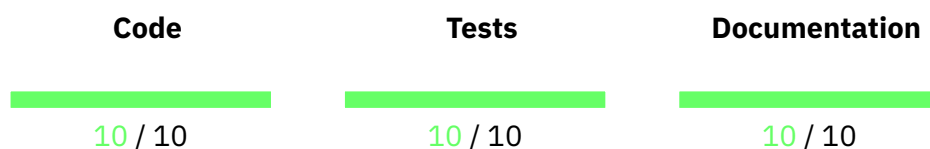
System Overview

DEIN Finance is a scalable platform that provides unlimited insurance options for users. It empowers users to buy or sell insurance for virtually anything, whether it's a physical or digital asset on any blockchain, a real-world or virtual event, or a custom and complex set of conditions.

DEIN offers a comprehensive solution that not only addresses the challenges of scalability, customization, and adaptability but also puts users in control of their risk exposure while providing the means to protect their funds.



Repository Coverage and Quality



Resonance's testing team has assessed the Code, Tests, and Documentation coverage and quality of the system and achieved the following results:

- The code follows development best practices and makes use of known patterns, standard libraries, and language guides. It is easily readable and uses the latest stable version of relevant components. Overall, **code quality is excellent**.
- Unit and integration tests are included. The tests cover both technical and functional requirements. Overall, **tests coverage and quality is excellent**.

- The documentation includes the specification of the system, technical details for the code, relevant explanations of workflows and interactions. Overall, **documentation coverage and quality is excellent.**

Target

The objective of this project is to conduct a comprehensive review and security analysis of the smart contracts that are contained within the specified repository.

The following items are included as targets of the security assessment:

- Repository: [dein-fi/dein-core/contracts](#)
- Hash: be0e9541720fea8b102edaf7b91cb3d95795351e

The following items are excluded:

- External and standard libraries
- Files pertaining to the deployment process
- Financial related attacks

Methodology

In the context of security audits, Resonance's primary objective is to portray the workflow of a real-world cyber attack against an entity or organization, and document in a report the findings, vulnerabilities, and techniques used by malicious actors. While several approaches can be taken into consideration during the assessment, Resonance's core value comes from the ability to correlate automated and manual analysis of system components and reach a comprehensive understanding and awareness with the customer on security-related issues.

Resonance implements several and extensive verifications based off industry's standards, such as, identification and exploitation of security vulnerabilities both public and proprietary, static and dynamic testing of relevant workflows, adherence and knowledge of security best practices, assurance of system specifications and requirements, and more. Resonance's approach is therefore consistent, credible and essential, for customers to maintain a low degree of risk exposure.

Ultimately, product owners are able to analyze the audit from the perspective of a malicious actor and distinguish where, how, and why security gaps exist in their assets, and mitigate them in a timely fashion.

Source Code Review - Solidity EVM

During source code reviews for Web3 assets, Resonance includes a specific methodology that better attempts to effectively test the system in check:

1. Review specifications, documentation, and functionalities
2. Assert functionalities work as intended and specified
3. Deploy system in test environment and execute deployment processes and tests
4. Perform automated code review with public and proprietary tools
5. Perform manual code review with several experienced engineers
6. Attempt to discover and exploit security-related findings
7. Examine code quality and adherence to development and security best practices
8. Specify concise recommendations and action items
9. Revise mitigating efforts and validate the security of the system

Additionally and specifically for Solidity EVM audits, the following attack scenarios and tests are recreated by Resonance to guarantee the most thorough coverage of the codebase:

- Reentrancy attacks
- Frontrunning attacks
- Unsafe external calls
- Unsafe third party integrations
- Denial of service
- Access control issues

- Inaccurate business logic implementations
- Incorrect gas usage
- Arithmetic issues
- Unsafe callbacks
- Timestamp dependence
- Mishandled panics, errors and exceptions



Severity Rating

Security findings identified by Resonance are rated based on a Severity Rating which is, in turn, calculated off the **impact** and **likelihood** of a related security incident taking place. This rating provides a way to capture the principal characteristics of a finding in these two categories and produce a score reflecting its severity. The score can then be translated into a qualitative representation to help customers properly assess and prioritize their vulnerability management processes.

The **impact** of a finding can be categorized in the following levels:

1. Weak - Inconsequential or minimal damage or loss
2. Medium - Temporary or partial damage or loss
3. Strong - Significant or unrecoverable damage or loss

The **likelihood** of a finding can be categorized in the following levels:

1. Unlikely - Requires substantial knowledge or effort or uncontrollable conditions
2. Likely - Requires technical knowledge or no special conditions
3. Very Likely - Requires trivial knowledge or effort or no conditions

		Likelihood		
		Very Likely	Likely	Unlikely
Impact	Strong	Critical	High	Medium
	Medium	High	Medium	Low
	Weak	Medium	Low	Info



Repository Coverage and Quality Rating

The assessment of Code, Tests, and Documentation coverage and quality is one of many goals of Resonance to maintain a high-level of accountability and excellence in building the Web3 industry. In Resonance it is believed to be paramount that builders start off with a good supporting base, not only development-wise, but also with the different security aspects in mind. A product, well thought out and built right from the start, is inherently a more secure product, and has the potential to be a game-changer for Web3's new generation of blockchains, smart contracts, and dApps.

Accordingly, Resonance implements the evaluation of the code, the tests, and the documentation on a score **from 1 to 10** (1 being the lowest and 10 being the highest) to assess their quality and coverage. In more detail:

- Code should follow development best practices, including usage of known patterns, standard libraries, and language guides. It should be easily readable throughout its structure, completed with relevant comments, and make use of the latest stable version components, which most of the times are naturally more secure.
- Tests should always be included to assess both technical and functional requirements of the system. Unit testing alone does not provide sufficient knowledge about the correct functioning of the code. Integration tests are often where most security issues are found, and should always be included. Furthermore, the tests should cover the entirety of the codebase, making sure no line of code is left unchecked.
- Documentation should provide sufficient knowledge for the users of the system. It is useful for developers and power-users to understand the technical and specification details behind each section of the code, as well as, regular users who need to discern the different functional workflows to interact with the system.

Findings

During the security audit, several findings were identified to possess a certain degree of security-related weaknesses. These findings, represented by unique IDs, are detailed in this section with relevant information including Severity, Category, Status, Code Section, Description, and Recommendation. Further extensive information may be included in corresponding appendices should it be required.

An overview of all the identified findings is outlined in the table below, where they are sorted by Severity and include a **Remediation Priority** metric asserted by Resonance's Testing Team. This metric characterizes findings as follows:

- ||||| "Quick Win" Requires little work for a high impact on risk reduction.
- |||| "Standard Fix" Requires an average amount of work to fully reduce the risk.
- ||| "Heavy Project" Requires extensive work for a low impact on risk reduction.

Findings ID	Description	Severity	Status
RES-01	Restake Rewards Allow Extra Benefits In Less Time		Acknowledged
RES-02	permit() Function Could Be Frontrun		Acknowledged
RES-03	Hardcoded blocks_per_day May Fail in L2 Chains or Consensus Upgrades		Resolved
RES-04	Missing Limits In setMinimumInsuranceCost()		Resolved
RES-05	Fee On Transfer Tokens May Not Work Properly		Acknowledged
RES-06	epoch in buyPolicy() Does Not Follow Documentation		Resolved
RES-07	Execution at Deadlines Should Be Allowed		Resolved
RES-08	TWAP Oracle Update Mechanism Cannot Be Left To Users		Resolved
RES-09	Potential TWAP Manipulation if DEIN/ETH Pool Has Low Liquidity		Acknowledged
RES-10	evidenceURI Could Contain Malicious Payloads		Resolved
RES-11	Non-external Function Names Should Begin With an Underscore		Resolved
RES-12	uint Should Not Be Initialized With 0		Resolved



Restake Rewards Allow Extra Benefits In Less Time

Medium RES-DEIN-INS01

Business Logic

Acknowledged

Code Section

- [contracts/DEINStaking.sol#L474](#)

Description

The `deinStaking.restakeReward()` function allows users to restake rewards in a new position or into an existing one. Restaking rewards in a existing position applies the multiplier from that staking position but without locking the tokens for that time. This way, users may obtain more rewards than expected, as they could obtain higher multipliers for less staking time. instead of staking in a new position.

Recommendation

To avoid extra rewards when restaking them, it is recommended to allow restaking rewards into new positions only.

Status

The issue has been acknowledged with the comment "This is the planned business logic. Users are only able to restake rewards they have accumulated through providing coverage, native staking, or liquidity staking. This means that even if they are restaking into an already existing position with a shorter locking period and a higher multiplier, they must have previously staked their tokens in other positions—putting themselves at risk—in order to earn those rewards."



permit() Function Could Be Frontrun

Medium

RES-DEIN-INS02

Data Validation

Acknowledged

Code Section

- `contracts/BMISCoverStaking.sol#L301`

Description

ERC-20 Permit is a modification of the standard ERC-20 token that allows users to delegate their spending rights to other addresses without having to send the tokens themselves or use ether to pay for gas for transactions. It uses the nonces mapping for replay protection. Once a signature is verified and approved, the nonce increases, invalidating the same signature being replayed.

When the user transaction is in the mempool, an attacker can take this signature, call the `token.permit` function on the token directly and make the user's transaction fail whenever it gets mined, causing a Denial of Service.

Recommendation

It is recommended to wrap `permit()` calls in try-catch blocks to continue execution if malicious users try to frontrun the transaction.

Status

The issue was acknowledged by Dein's team. The development team stated "contract is out of our audit scope".



Hardcoded blocks_per_day May Fail in L2 Chains or Consensus Upgrades

Low

RES-DEIN-INS03

Data Validation

Resolved

Code Section

- `contracts/Globals.sol#L163`

Description

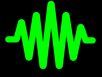
Hardcoding `blocks_per_day` in a smart contract can lead to several issues, mainly because the actual block time on Ethereum can fluctuate. This may lead to inaccurate time calculations, incorrect rewards and interest or incompatibility across L2 chains.

Recommendation

It is recommended to implement a setter function that allows the admin to update the value of `blocks_per_day`.

Status

The issue has been fixed in commit `1ce2ce68231967c208c82a1002c72a5d2851e1ff`.



Missing Limits In `setMinimumInsuranceCost()`

Low

RES-DEIN-INS04

Data Validation

Resolved

Code Section

- [contracts/PolicyQuote.sol#L197](#)

Description

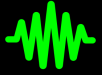
The `setMinimumInsuranceCost()` function does not limit the insurance cost. This may affect the user experience and it may not be practical.

Recommendation

It is recommended to limit the minimum cost to an appropriate value.

Status

The issue has been fixed in commit [1ce2ce68231967c208c82a1002c72a5d2851e1ff](#).



Fee On Transfer Tokens May Not Work Properly

Low

RES-DEIN-INS05

Business Logic

Acknowledged

Code Section

- `contracts/CapitalPool.sol#L653`

Description

Fee-on-transfer tokens are ERC-20 tokens that deduct a fee whenever they are transferred. DEIN protocol uses USDT as stablecoin but it does not implement controls to cover this fee on transfer token if fee is set. This may lead to accounting errors that will break the protocol.

Recommendation

It is recommended to implement controls to calculate the transferred amounts if case fees are applied to the tokens.

Status

The issue was acknowledged by Dein's team. The development team stated "In the event that it does happen, there will be a prior announcement, and we will be able to address and resolve it promptly at that point".



epoch in buyPolicy() Does Not Follow Documentation

Low

RES-DEIN-INS06

Data Validation

Resolved

Code Section

- [contracts/PolicyBookFacade.sol#L221](#)

Description

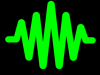
The documentation and the frontend show that the insurance duration should be between 1 and 15 epoch but it is possible to call buyPolicy with a higher value.

Recommendation

Control measures should be implemented to limit the epochs submitted by users according to the documentation.

Status

The issue was addressed by Dein's team. The development team stated "good point for FE side".



Execution at Deadlines Should Be Allowed

Low

RES-DEIN-INS07

Data Validation

Resolved

Code Section

- [contracts/PolicyQuote.sol#L446](#)

Description

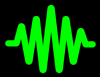
`validToTime` in the `_isDiscountModelActive()` function specifies whether a discount is applied until the specified date, but it may also be applicable in the scenario where `block.timestamp` is equal to `validToTime`.

Recommendation

It is recommended to consider this scenario and update the returned value in case the last moment of the discount should be covered as well.

Status

The issue has been fixed in commit [1ce2ce68231967c208c82a1002c72a5d2851e1ff](#).



TWAP Oracle Update Mechanism Cannot Be Left To Users

Low

RES-DEIN-INS08

Data Validation

Resolved

Code Section

- `contracts/helpers/PriceFeed.sol#L171`

Description

The `UniswapV2OracleLibrary` is used to calculate prices based on the Uniswap V2 cumulative price, so contracts must manually call functions that capture the `priceCumulativeLast`.

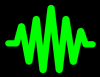
Discontinuous reporting may break the concept of TWAP and lead to unequal weightings to certain data points or price mismatch in case prices fluctuate rapidly.

In case of low activity, the protocol updates the prices automatically every 3 days, but this time may not be enough.

Recommendation

Status

The issue has been fixed by the team in commit id `1ce2ce68231967c208c82a1002c72a5d2851e1ff`. They mention that this issue will be handled by a serverless cron job.



Potential TWAP Manipulation if DEIN/ETH Pool Has Low Liquidity

Low

RES-DEIN-INS09

Data Validation

Acknowledged

Code Section

- [contracts/helpers/PriceFeed.sol#L171](#)

Description

if the DEIN/ETH pool has low liquidity, it is easier for a malicious user to manipulate the price with less capital. This user could place a large trade to push the price in one direction and then reverse it before TWAP correctly records the average.

Recommendation

Some recommendations for address this issue may be

- check amount of liquidity pool used as price source.
- Good liquidity for the involved tokens across different markets to facilitate arbitrage.
- Monitoring of liquidity over time.

More info on [TWAP Manipulation](#).

Status

The issue was acknowledged by Dein's team. The development team stated "Given this architecture, we believe the issue does not pose a real threat to our platform at this time".



evidenceURI Could Contain Malicious Payloads

Info

RES-DEIN-INS10

Data Validation

Resolved

Code Section

- [contracts/PolicyBookFacade.sol#L445](#)

Description

The evidenceURI added during claims is not sanitized and could allow room for json injection attacks or bad chars that could affect the web2 infrastructure.

Recommendation

It is recommended to add sanitization to evidences when presented in the frontend.

Status

The issue has been fixed in commit [1ce2ce68231967c208c82a1002c72a5d2851e1ff](#).



Non-external Function Names Should Begin With an Underscore

Info

RES-DEIN-INS11

Code Quality

Resolved

Code Section

- [contracts/PolicyQuote.sol#L446](#)

Description

According to the Solidity Style Guide, non-external function names should begin with an underscore.

Recommendation

Consider renaming the affected private functions.

Status

The issue has been fixed in commit [1ce2ce68231967c208c82a1002c72a5d2851e1ff](#).



uint Should Not Be Initialized With 0

Info

RES-DEIN-INS12

Gas Optimization

Resolved

Code Section

- [contracts/YieldGenerator.sol#L265](#)

Description

Initializing variables to their default values spends gas unnecessarily.

Recommendation

It is recommended to not initialize variables to their default values, in this case 0.

Status

The issue has been fixed in commit [1ce2ce68231967c208c82a1002c72a5d2851e1ff](#).

Proof of Concepts

No Proof-of-Concept was deemed relevant to describe findings in this engagement.